BeClone: Behavior Cloning with Inference for Real-Time Strategy Games

Derek Martin and Arnav Jhala
North Carolina State University
Raleigh, NC 27606

Abstract

Behavior cloning (BC) techniques that combine self-play capabilities with imitation learning from experts to refine self-play models have shown performance improvement in robotics simulation domains. In this paper, we investigate the performance of this technique on Real-time strategy game tasks. One challenge with this approach is the training time of agents and real-time adaptation to opponent strategies. We present a framework BeClone for training agents in two phases. The first phase uses behavior cloning (BC) to learn base policies. The second phase uses an advantage actor-critic (A2C) reinforcement learning algorithm to adapt base strategies through self-play to explore the action space. We demonstrate the success of the BeClone framework on the microRTS domain through the comparison of the performance of the agents against the baseline A2C agent proposed by Huang and Ontanon. Our results on the resource gathering benchmark show improvement in agent performance both in terms of rewards and training time.

Introduction

Modern video games provide researchers with a complex and diverse environment to develop and test agent models and algorithms. Real-time strategy (RTS) games have been an important genre of games for current AI researchers due to characteristics like partial-observability, simultaneous play, and reasoning across multiple competencies. Unlike in turn-based games like Chess or Go, the time to take action is an important factor in building successful agents. Reinforcement learning algorithms that have been successful in developing agents for turn-based games face challenges in RTS domains. Partial observability of the game world also increases complexity of agent models because strategies learned during limited self-play simulations are not sufficient in real-time reactions to opponent strategies. These challenges for RTS games have been well-documented in recent years [14].

To overcome these challenges, a combination of heuristic search and action policy learning algorithms have been utilized. Heuristic search algorithms have an authoring bottleneck in terms of fine tuning heuristics and definitions of action operators for planning algorithms. Portfolio search approach [8, 7] that utilizes an ensemble of heuristics to determine best actions has shown promise in search-based methods for RTS agent modeling. There has been initial work in addressing the authoring bottleneck of search-based algorithms in RTS games by learning hierarchical task network like structures [17]. This paper will focus on reinforcement learning, specifically deep reinforcement learning models that have been recently utilized to model various competencies in RTS game playing agents [10, 20]. For RL algorithms in this domain, two hurdles have been sparse rewards and exploring undesirable or sub-optimal states. RL algorithms for RTS agents need efficient representations and directed exploration algorithm to improve performance [20, 12]. This paper describes a two phase framework that utilizes imitation learning, specifically behavior cloning, to guide the agent towards a desirable base policy by observing of expert game replays and, in the second phase, the agent utilizes the advantage actor-critic (A2C) framework to learn adaptations to the base policy to maximize rewards through self-play. Alone, pure imitation learning can only perform as well as the experts it observes and pure RL, as previously stated, struggles with sparse rewards and exploring undesirable states. We use the global state representation proposed by Huang and Ontanon (HA) [20] to train an agent with the proposed framework. Our results on the benchmark domain show improvement both in terms of rewards and learning time (with regard to the first successful harvest and return of resources).

We present experimental results on the μRTS domain. μRTS is an open-source simplified RTS game that implements the key challenges of RTS games and was designed specifically as an open sandbox for AI research [18]. It allows researchers to configure the game environment for their experiments. Figure 1 shows a frame from μRTS. The green squares are resources that the worker (gray circles) units gather to return the base (white squares). The numbers on the units indicates the number of resources that the unit contains. After Workers have gathered enough resources, they can build barracks (gray squares) that produce light, heavy, or ranged units. These units share a relationship similar to rock, paper, scissors where light units work best against ranged, ranged work best against heavy, and heavy work best

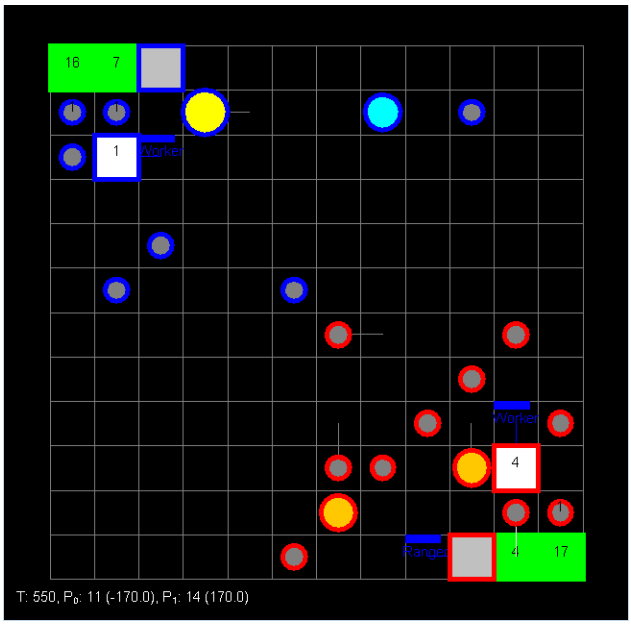

Figure 1: Blue outline on units indicates player 2 and red outline indicates player 1. Squares indicates stationary units and circles indicate movable units. Green squares are resources, white squares are bases, gray squares are barracks, small gray circles are worker units, yellow circles are light military units, blue circles are ranged military units, and orange circles are heavy military units.

against light [21]. Our experiments show that our approach does significantly better on the small, 4x4 map and slightly better or the same on the larger, 6x6 and 8x8 maps.

Background

Real-time Strategy Games: In real-time strategy games, players control armies composed of a variety of units, usually in an partially observable environment. Partially observable in the sense that the player does not have the ability to continuously monitor opponents actions to predict their strategy. Some RTS games also limit how much of the map the player can see(Fog of War). These factors (real-time, multi-agent, and partially observable) combined make RTS games challenging for humans and learning agents [6, 20]. The real-time factor forces the player to be quick when making decisions because players can make actions simultaneously and actions have different durations. Not only do players have a short amount of time to make decisions, but they also have a large action and state space to consider when making those decisions. Player build their armies based upon some plan or strategy causing their action space to grow, or branch, exponentially. For example, 10 units with 5 possibles actions results in a branching factor of $\approx$10 million [14]. Finally, since RTS games are usually partially observable, this means players have to work under uncertainty.

RL in RTS: Reinforcement learning is a good fit to model several competencies within RTS games such as spatial reasoning [10]. In Sharma et al.[3], they combine reinforcement learning with case-based reasoning (CBR) in the MADRTS game domain. They present a multilayered architecture that focuses on different competencies of the agent, and residing in the middle of this architecture is a case-based reinforcement learner that learns to make tactical decisions over the game's action space. Their experiments showed that their learning agent not only succeeded on individual tasks, but also had significant performance gains when using knowledge transferred from previous tasks. This work shows the effectiveness of using competency specific expert agents and reinforcement learning to perform well in real-time strategy games. One challenge in this work is the time to train the agent on individual competencies independently and then tuning the case-based module to determine the policy of prioritizing recommendations from each module to make the final decision. As shown by Weber et al [5], this agent architecture leads to agents that struggle with maintenance of a coherent plan due to constant queries and responses to the case base.

Similar to Sharma et al. [3], Lee et al. [4] use case-based reasoning (CBR) with reinforcement learning, but they use it in the RTS game WARGUS. This system uses CBR to fulfill all the requirements necessary for reinforcement learning to operate in RTS games. Those requirements being: (1) a state space that can be represented as an Markov Decision Processes, future states depend on the actions made in previous states; (2) an abstracted state space that is reduced and guarantees real-time performance, and (3) an online state space that allows the agent to use information about the game state. The state space and cases are built by learning player behaviors then using this knowledge to simplify the state space and make decisions. Their experiments show that their method outperforms the dynamic, adapting human-like scripts for the WARGUS. This is another example of how useful transfer learning through observing other or previous agents is when using reinforcement learning in RTS games.

The work that is most relevant to this paper is by HA [20]. They use deep reinforcement learning in μRTS to learn the best state representation for training an RL agent for the resource gathering competency. They compare two types of representations: global and local. Global representation is focused on training the agent to select both the unit and its action. The global representation restricts the agent to one action at a time. The local representation focused on individual units in the game that are controlled by the agent. Each unit learns what action perform for itself. They demonstrate their algorithm on three maps (4x4, 6x6, and 8x8), and report that local representation performs better global representation. They also report a significant impact of map size on agent performance. They believe this is due to the navigation challenges added by the larger map. Their agent struggles on larger maps and control of larger number of units due to sparsity in rewards for the RL algorithm. This indicates that more efficient representations are needed to improve the agent's performance on the resource gathering task. We show how a two step algorithm which performs imitation learning to learn base policies from human gameplay observations followed by a self-play advantage actor-critic

algorithm (A2C) for fine-tuning the base policies improves performance of the RL agent on this resources gathering benchmark.

Imitation Learning from Observation: Imitation learning is the process of an agent learning how to perform a task by observing an expert first then trying to imitate the expert. One of the earliest studies using imitation learning is Abbeel and Ng's [2]'s work on an apprenticeship, another name for imitation learning, agent for learning the reward functions for Gridworld and a driving simulator. In this study they characterize how long it takes for an apprentice agent to converge and perform similar to the expert agents that it observes. For the Gridworld experiments, finding a small set of cells with positive rewards and varying the number of experts sampled, they found that both versions of their apprentice agent in the first experiment converged similar to the observed expert agents. In the second Gridworld experiment they compare the two previous apprentice agents to three other agents (two parameterized agents and one "mimic the expert" agent) and find that their agents outperformed the other three agents. For the second, car driving simulation experiment, Abbeel and Ng [2] used imitation learning to try teach their agent different driving styles. After letting the agent observe a number of "expert" driving styles for approximately 2 minutes, the agent attempts to learn a policy that best approximates each driving style. Their results show that the agent is able to successfully mimic each driving style.

Torabi et al. [19] propose a BC from observations framework. In this framework their agent uses an inverse dynamic model to learn how actions effect the environment and behavior cloning. The agent first interacts with the environment to gather prior experience/base understanding of the environment then uses expert state-demonstrations only (no expert actions are provided) to improve its model. They ran experiments in the OpenAI gym and compared their framework against three other imitation learning algorithms: BC, Feature Expectation Matching (FEM), and Generative Adversarial Imitation Learning. Their approach is the only one that does not use the demonstrator's actions while training, and they found that their framework works as well or better than the methods that require access to the demonstrator's actions. We based on BC approach on this approach, except we included the action because our goal is a specific task. When we scale this research to a full game, we plan to remove access to the demonstrator's actions or use self-supervised imitation learning to ensure our agent is more generalized.

Imitation learning has also be done through observations from video, like Aytar et al.'s [15] study. In this study, their agents learn how to play three games, Montezuma's Revenge, Pitfall! and Private Eye, with sparse rewards by observing YouTube videos of humans playing those games. They found that pure RL agents are not able to collect the sparse rewards in Montezuma's Revenge and Pitfall! and are only able to gather the first two rewards in Private Eye. A key takeaway from this work is that their approach uses a combination of rewards from multiple modalities (video and audio) in order to address the limitation of sparse rewards.

Global Representation

Features	Possible Values
Hit Points	0,1,2,3,4,5, $\geq$6
Resources	0,1,2,3,4,5, $\geq$6
Owner	Player 1,-,Player 2
Unit Type	-,resource,base,barracks,worker,light,heavy
Action	-,move,harvest,return,produce,attack

Table 1: Features used for the global representation.

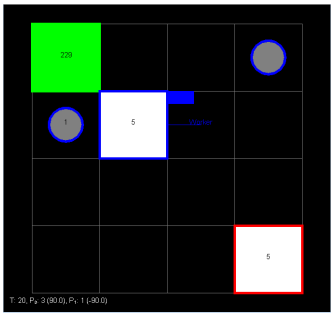

Figure 2: Harvest opportunity. The Worker unit has 1 resource and is near the base, so the agent wants to select this unit and tell it to return the resource to the base.

Approach

We present a framework that trains an agent on the resource gathering benchmark in μRTS domain. This algorithm operates in two phases. In the first phase, the agent learns a base policy based on gameplay observations from a hard-coded agent. In the second phase, the agent refines the base policy through the advantage actor-critic algorithm with self-play simulations. We treat the game map as a grid and each feature map conforms to this grid. Each cell of the feature map represents some information about the current game state, such as: resources held, team, etc. The full table of features and associated values are located in Table 1 above. Imitation Learning is used to create a baseline policy to be used later for reinforcement learning. The first half of training on observation of gameplay of expert agents. These expert agents could be human agents such as the ones used by Weber et al. [6] or hand-programmed planning agents with different heuristic functions for demonstrating a variety of behaviors. This informs the agent of possible actions to take in given game states and basic policies of resource gathering.

Global Representation: The environment is represented as an observation matrix with a size of: width (w) x height (h) x number of features (nf). There are 5 feature maps in total that keep track of hit points, resources held, team, action, and parameter. Each feature map has the same size as the game map resulting in the observation matrix size. To complete the input tensor, each cell of the feature maps contains a vector with the size of the largest feature (nc). In this paper, the largest feature has 7 values, so we use this for each cell. The final hot-encoded tensor/vector has a size of:

$nf * w * h * nc$. This is the same representation that HA uses in their experiments.

This representation is used to select the unit to control and the action for the selected unit to perform. Similar to HA [20], we only focus on harvesting resources, so the only actions we focus on are *Wait* (No Operation), *Move*, *Harvest*, and *Return*. Some of the actions require a parameter, such as direction. For example, if the unit, in the first row, in figure 2 wants to harvest resources then the model needs to output *Harvest* and *left* as the action and parameter values. The agent also has predicts the location of the unit it wants to control.

As a result, the action vector is: $[a_t^x, a_t^y, a_t^{action}, a_t^{param}]$. Where, a_t^x is the predicted unit x-coordinate, a_t^y is the predicted unit y-coordinate, a_t^{action} is the selected action in frame t, and a_t^{param} is the parameter of the selected action. The agent learns which moves are valid and invalid for the selected unit. Also, the agent is only able to make one command per turn. To summarize, the global representation encodes the entire game state, and the RL agent uses this information to select what unit to control, what action to perform, and the direction to perform the action.

Advantage Actor Critic: The advantage actor-critic (A2C) algorithm is a reinforcement learning algorithm for learning generalized policies. We chose this framework because it enables the use of distributed RL by utilizing more than one actor and a critic, and we could focus on adapting to local contexts such as risk management [16]. However, for this paper we only focus on resource harvesting to understand the effects of directing the exploration policy of our RL agent using imitation learning. We base our implementation on Mnih et al.'s [13] A3C algorithm to implement our A2C algorithm similar to HA [20]. The A2C algorithm has two components: the actor and the critic. The actor maintains the policy and handles the decision making, while the critic criticizes (calculates the loss to update the actor's policy) the actor's decisions and estimates the value function of the environment, and they work together to maximize the expected reward [1]. After every episode, t_{max} or a terminal state is reached, the policy and value function are updated. The policy is updated via: $\nabla_{\theta'} log\pi(a_t|s_t;\theta')A(s_t,a_t;\theta,\theta_v)$ where $A(s_t,a_t;\theta,\theta_v)$ is the estimated advantage function calculated by $\Sigma_{i=0}^{k-1}\gamma^i r_{t+1} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v)$, where k represents the number of states visited during the episode; has a max of t_{max}.

Behavior Cloning: Behavior cloning (BC) trains an agent using data collected from expert playthroughs (human or other agents). BC with inference means while training the agent tries to infer the expert agent's selected action given a sampled state and its resulting state (outcome of the expert's action). Our hypothesis with BC was if we let our agent observe how expert agents harvest resources, it would give a baseline policy to use when focusing on harvesting resources or any other task that we provide samples for. Using BC, we provide demonstrations for our agent to direct its exploration towards important areas of the action space and form a baseline policy. Directing its exploration also actively helps to minimize the chance of the agent exploring undesirable states [12]. The resulting policy gives the agent a baseline, and if trained long enough would eventually perform exactly like the experts unlike using a random exploration algorithm like most RL algorithms [12, 2].

As mentioned earlier, reinforcement learning struggles in environments with sparse rewards, and learning how to harvest from scratch means the agent will experience sparse rewards as it learns how to harvest by itself. This exploration for resources also gets combined with learning navigation on the grid with respect to the relative locations of the base and resources. Starting with a learned player model the agent learns the basic movements and navigation primitives. There are some limitations of using the learned player models. If the observed agents are not sufficiently varied in terms of their diversity of play then the agent could learn biases of the observed agents. Beyond the μRTS domain, the learning agent would also be sensitive to the variety and size of maps in which the observed agents are performing. Also, we are only concerned with navigation and harvesting tasks in this work but as the player model takes into account all available actions that include additional units like production, attack, and defense(*Wait*, *Move*, *Return*, *Harvest*, *Produce*, and *Attack*) it becomes more challenging for the agent to learn coherent policies due to interleaved actions. This challenge has previously been documented and addressed to some extent in the StarCraft community [9, 11].

Training

Training through Behavior Cloning: As previously stated, the first half of BeClone's training involves BC using the hard coded agents supplied by Ontañón et al. with the μRTS source code. We used the WorkerRush and WorkerDefense agents to learn a model. The reason we chose these agents is they have at least one worker unit dedicated to harvesting resources as quickly as possible. Therefore, our agent only needs to observe the game state then attempt to infer the action of the worker harvesting resources. While training with BC, our A2C model is treated as an ordinary feed forward neural network. The hardcoded agent's actions are used as labels to produce the network error, then those values are backpropagated to update the weights (treated as the policy) of the networks. The agents (WorkerRush and WorkerDefense) are trained against the Idle agent in the μRTS tournament environment for 100 iterations (2000 timesteps or terminal game state), so BeClone can focus solely on harvesting resources without having to worry about being attacked.

Training through Reinforcement Learning: After learning a baseline policy through BC, we utilize A2C to complete the rest of the training. We transferred the weights from the model in the previous section to initialize the full network then we followed the implementation of HA for updating the network. After initialization, the observation tensor is fed into the network policy network and value network (see Figure 3 for network architecture). The value network takes the observation tensor and outputs an estimated valuation for the current state. When the observation tensor is fed into the policy network, the data is split into 4 logits

Parameter Names	Values
Maps	4x4, 6x6, 8x8
Learning Rate of Adam Optimizer	0.0007
Random Seed	1,2,3
Episode Length	2,000 time steps
Total Time Steps	2M time steps
γ (Discount Factor)	0.99
β (Value Funcction Normalization)	0.25
η (Entropy Regularization Coefficient)	0.1
ω (Gradient Norm Threshold)	0.5

Table 2: Hyperparameters used for training.

vectors. Each vector is then converted into a probability distribution for sampling values to produce the action vector a = $[a_t^x, a_t^y, a_t^{action}, a_t^{param}]$.

To calculate the loss over the episode, we use the formula:

$$\begin{aligned} log\pi_\theta(a_t|s_t) = & log\pi_\theta(a_t^x|s_t) \\ & + log\pi_\theta(a_t^y|s_t) \\ & + log\pi_\theta(a_t^{action}|s_t) \\ & + log\pi_\theta(a_t^{param}|s_t) \end{aligned} \tag{1}$$

s_t is the state at time step t and a_t is the action probability. To compute entropy we use the formula:

$$entropy = \sum_a \pi_\theta(a|s_t) log\pi_\theta(a|s_t) \tag{2}$$

this is the sum of the entropy values $a^x, a^y, a^{action}, a^{param}$.
To calculate the error, or gradient, of the network:

$$\begin{aligned} & \underbrace{(G_t - v_{\theta'}(s_t))\nabla_\theta log\pi_\theta(a_t|s_t)}_{policy\ gradient} + \\ & \underbrace{\beta((G_t - v_{\theta'}(s_t))\nabla_{\theta'} v_{\theta'}(s_t)}_{value\ estimation\ gradient} + \\ & \underbrace{\eta \sum_a \pi_\theta(a|s_t) log\pi_\theta(a|s_t)}_{entropy\ regularization} \end{aligned} \tag{3}$$

Here θ' and θ are the weights for the value network and the policy network, β and η are hyperparameters for controlling the value estimation and entropy regularization gradients, and G_t is the discounted rewards. We used the same parameters as Huang and Ontañón, and you can find the full list of parameters in Table 2.

Experiment

In this section we will compare our approach to HA's approach, on the same maps.

Setup: We used similar maps to HA. We made the 4x4, 6x6, and 8x8 maps shown in Figure 4. They consist of two worker units, two bases (player's base and the enemy base), and one resource unit containing 230 resources. The reason behind the resource block containing 200+ resources is because, ideally, over the 2000 time step episode the two worker agents could harvest at most 200 resources on the 4x4 map. Each action takes 10 time steps to complete, so to harvest and return one resource each worker would need to harvest a resource (10 steps) and return the resource (10 steps) for a total of 20 time steps. Over the whole episode (2000 time steps) that would more the two agents would be able to harvest 200 resources: 2*2000/20 = 200.

Comparison: BeClone, when compared to HA, almost doubled the average reward on the 4x4 map, had a slight increase in average reward on the 6x6 map, and no change on the 8x8 map. Another notable outcome is not only did BeClone increase performance, it significantly decreased the harvest and return times. For the 4x4 map, there was a significant increase in average reward but also a significant increase in the average time taken for harvesting and returning resources. This may be an acceptable trade-off because even though it takes longer on average, it also performs almost twice as well as HA's harvesting agent. However, for the larger maps (6x6 and 8x8), our approach significantly decreased time taken to harvest and return resource (when the agent was successfully able to return the resource). There was no significant increase in the average reward for these maps though.

This improvement was a result of imitation learning. Imitation learning improved HA's global representation agent because BeClone observed an "expert" and learned what to do in certain game states where as the base agent had to learn what to do through trial and error. Also, if given enough time to train, BeClone would eventually learn the hardcoded agents' behaviors. This gave BeClone a great baseline policy to operate with. Then using A2C enabled the agent to try to maximize its performance. However, during the reinforcement learning phase of training and testing, we noticed that the agent would often move the worker units to the base or try to return resources prior to harvesting a resource. This is why we believe that BeClone performed roughly the same as HA's global agent on the larger maps. We believe if we influence the reward structure such that harvesting a reward weighs more than returning the reward or adding negative rewards when the agent attempts to return a resource without a resource first that this could potentially improve BeClone's performance on the larger, 6x6 and 8x8, maps.

Conclusions and Future Work

Our results show that BeClone made a significant improvement on the smaller map and similar on the larger maps to previously best performing agents on the benchmark [20]. These reults indicate that BC is promising to consider as an effective addition to the A2C model. Without shaping the reward structure or invalid action masking, we believe that the BC model on the larger maps learned the path to and from the base and showed improved performance on

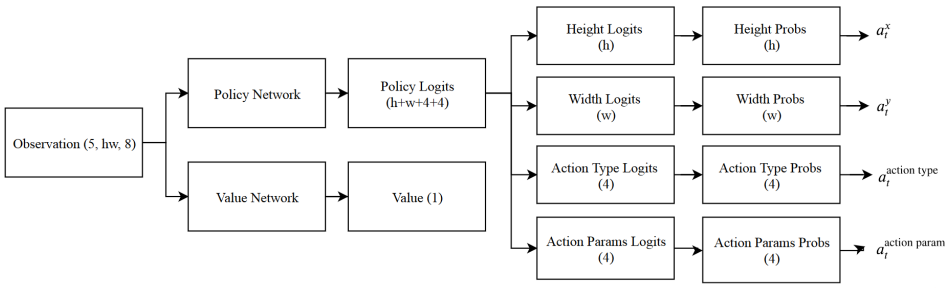

Figure 3: Network architecture.

	Map	$t_{first\ harvest}$	$t_{first\ return}$	r
HA	4x4	99.0	167.33	13.13
BeClone	4x4	233.33	470.67	25.98
HA	6x6	533.33	1931.20	.07
BeClone	6x6	155.71	834.00	.1
HA	8x8	1464.53	-	0.00
BeClone	8x8	656.00	-	0.00

Table 3: Comparison of HA's global approach versus our global approach.

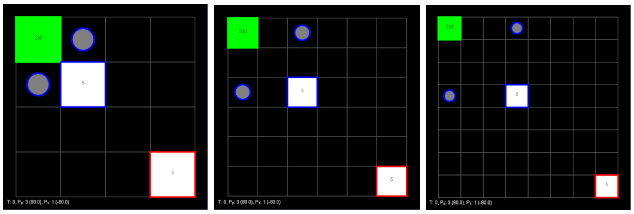

Figure 4: Maps used in experiment

resource gathering task. It did struggle on the larger map in successfully returning the resources to the base, similar to prior work. Often we noticed that BeClone would move one or both worker units to the base and would not move them too far away from the base. Initially one of the issues with the agents in the original formulation of the task was that the rewards were not set up in a way that harvesting and navigation tasks had to be combined to encourage efficient navigation with respect to timely harvesting and delivery of resources back to the base. In the original setup, the agent has no punishment for returning back to the base without a resource leading to this inefficiency in the learned agent. In the imitation learning phase, the reward to correct movement and delivery are connected to the replication of example movements of observed bots that are more efficient in combining movement with timely and correct delivery of resources. This is an important insight for future work.

In summary, we have introduced a framework that combines BC with actor-critic reinforcement learning to improve performance over current benchmarks of resource gathering tasks in RTS game environments. Immediate next steps for this work are in scaling up learning in terms of domain and task complexity. Instead of directly cloning behavior, using cloned behavior tasks as a partial plan and inferring actions that are appropriate for the current context is a direction worth exploring. On the BC part, there are several interesting avenues to pursue on the formal characteristics of quantity and diversity of examples, learning from a variety of human traces, difference between observations from experts vs. novice players, and learning from state observation.

References

[1] Vijay R Konda and John N Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems*. 2000, pp. 1008–1014.

[2] Pieter Abbeel and Andrew Y Ng. "Apprenticeship learning via inverse reinforcement learning". In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 1.

[3] Manu Sharma et al. "Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL." In: *IJCAI*. Vol. 7. 2007, pp. 1041–1046.

[4] Jaeyong Lee, Bonjung Koo, and Kyungwhan Oh. "State space optimization using plan recognition and reinforcement learning on RTS game". In: *Proceedings of the International Conference on Artificial Intelligence, Knowledge Engineering, and Data Bases*. 2008.

[5] Ben G Weber, Michael Mateas, and Arnav Jhala. "Case-based goal formulation". In: *Proceedings of the AAAI Workshop on Goal-Driven Autonomy*. 2010.

[6] Ben George Weber, Michael Mateas, and Arnav Jhala. "Building Human-Level AI for Real-Time Strategy Games." In: *AAAI Fall Symposium: Advances in Cognitive Systems*. Vol. 11. 2011, p. 01.

[7] David Churchill and Michael Buro. "Incorporating search algorithms into RTS game agents". In: *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*. Citeseer. 2012.

[8] David Churchill, Abdallah Saffidine, and Michael Buro. "Fast heuristic search for RTS game combat scenarios". In: *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*. Citeseer. 2012.

[9] Ben George Weber, Michael Mateas, and Arnav Jhala. "Learning from Demonstration for Goal-Driven Autonomy." In: *AAAI*. 2012.

[10] Michael A Leece and Arnav Jhala. "Reinforcement learning for spatial reasoning in strategy games". In: *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2013.

[11] Michael A Leece and Arnav Jhala. "Sequential pattern mining in starcraft: Brood war for short and long-term goals". In: *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2014.

[12] Javier Garcıa and Fernando Fernández. "A comprehensive survey on safe reinforcement learning". In: *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480.

[13] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. 2016, pp. 1928–1937.

[14] Santiago Ontanón. "Combinatorial multi-armed bandits for real-time strategy games". In: *Journal of Artificial Intelligence Research* 58 (2017), pp. 665–702.

[15] Yusuf Aytar et al. "Playing hard exploration games by watching youtube". In: *Advances in Neural Information Processing Systems*. 2018, pp. 2930–2941.

[16] Lasse Espeholt et al. "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1407–1416.

[17] Michael Ong Leece. "Learning Hierarchical Abstractions from Human Demonstrations for Application-Scale Domains". PhD thesis. UC Santa Cruz, 2018.

[18] Santiago Ontañón et al. "The first microrts artificial intelligence competition". In: *AI Magazine* 39.1 (2018), pp. 75–83.

[19] Faraz Torabi, Garrett Warnell, and Peter Stone. "Behavioral cloning from observation". In: *arXiv preprint arXiv:1805.01954* (2018).

[20] Shengyi Huang and Santiago Ontañón. "Comparing Observation and Action Representations for Deep Reinforcement Learning in MicroRTS". In: *arXiv preprint arXiv:1910.12134* (2019).

[21] Florian Richoux. "microPhantom: Playing microRTS under uncertainty and chaos". In: *arXiv preprint arXiv:2005.11019* (2020).